# Architecture

The User Services Platform consists of a collection of Endpoints (Agents and Controllers) that allow applications to manipulate Service Elements. These Service Elements are made up of a set of Objects and parameters that model a given service, such as network interfaces, software modules, device firmware, remote elements proxied through another interface, virtual elements, or other managed services.

USP is made up of several architectural components:

* Mechanisms for discovery and trust establishment
* Transport of messages over one or more Message Transfer Protocols (MTPs) with associated MTP security
* A set of standardized messages based on the CRUD model (create, read, update, delete), plus an object defined operations mechanism and an notification mechanism (CRUD-ON).
* A method for encoding messages for transport
* Authorization and access control on a per element basis
* A method for modeling service elements using a set of objects, parameters, operations, and events (supported and instantiated data models)

## Endpoints

A USP endpoint can act as Agent or a Controller. Controllers only messages to Agents, and Agents send messages to Controllers. A USP Endpoint communicates over a secure session between other endpoints, over one or more Message Transfer Protocols (MTP) that may or may not be secured.

Figure ARC.1 - USP Agent and Controller Architecture

### Agents

A USP Agent exposes (to Controllers) one or more Service Elements that are represented in its data model. It contains or references both an Instantiated Data Model (representing the current state of Service Elements it represents) and a Supported Data Model.

### Controllers

A USP Controller manipulates (through Agents) a set of Service Elements that are represented in Agent data models. It may maintain a database of Agents, their capabilities, and their states, in any combination. A Controller usually acts as an interface to a user application or policy engine that uses the User Services Platform to address particular use cases.

### Endpoint Identifier

Endpoints are identified by an Endpoint Identifier.

The Endpoint Identifier is a locally or globally unique USP layer identifier of an Endpoint. Whether it is globally or locally unique depends on the scheme used for assignment.

The Endpoint Identifier (ID) is used in Message Headers and various Parameters to uniquely identify Controller and Agent Endpoints. It can be globally or locally unique, either among all Endpoints or among all Controllers or all Agents, depending on the scheme used for assignment.

The Endpoint ID is comprised of two mandatory and one optionally mandatory components: authority-scheme, authority-id, and instance-id.

These three components are combined as:

authority-scheme ":" [authority-id] ":" instance-id

The format of the authority-id is dictated by the authority-scheme. The format of the instance-id is dictated either by the authority-scheme or by the entity identified by the authority-id.

An Endpoint ID can be expressed as a urn in the bbf namespace as

"urn:bbf:usp:id:" authority-scheme ":" [authority-id] ":" instance-id

#### authority-scheme and authority-id

The authority-scheme follows the following syntax:

authority-scheme = "oui" | "cid" | "pen" | "self" | "user" | "os" | "ops" | "uuid" | "imei" | "proto" | "doc"

How these authority-scheme values impact the format and values of authority-id and instance-id is described below.

The authority defined by an OUI, CID, or Private Enterprise Number (including OUI used in "ops" and "os" authority scheme) is responsible for ensuring the uniqueness of the resulting Endpoint ID. Uniqueness can be global, local, unique across all Endpoints, or unique among all Controllers or all Agents. For the "user" authority scheme, the assigning user or machine is responsible for ensuring uniqueness. For the "self" authority scheme, the Endpoint is responsible for ensuring uniqueness.

**R-ARC.0** - A Controller and Agent within the same ecosystem MAY use the same Endpoint ID.

**R-ARC.1** - Endpoints MUST tolerate the same Endpoint ID being used by an Agent and a Controller in the same ecosystem.

**R-ARC.2** - Endpoints that share the same Endpoint ID MUST NOT communicate with each other via USP.

No conflict identification or resolution process is defined in USP to deal with a situation where an Endpoint ID is not unique among either all Agents or all Controllers in whatever ecosystem it operates. Therefore, a non-unique Endpoint ID will result in unpredictable behavior. An Endpoint ID that changes after having been used to identify an Endpoint can also result in unpredictable behavior.

Unless the authority responsible for assigning an Endpoint ID assigns meaning to an Agent and Controller having the same Endpoint ID, no meaning can be construed. That is, unless the assigning authority specifically states that an Agent and Controller with the same Endpoint ID are somehow related, no relationship can be assumed to exist.

|  |  |
| --- | --- |
| authority-scheme | usage and rules for authority-id and instance-id |
| oui | authority-id MUST be an OUI assigned and registered by the [IEEE Registration Authority](https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries) to the entity responsible for this Endpoint. authority-id MUST use hex encoding of the 24-bit ID (resulting in 6 hex characters). instance-id syntax is defined by this entity, who is also responsible for determining instance-id assignment mechanisms and for ensuring uniqueness of the instance-id within the context of the OUI. Example:oui:00256D:my-unique-bbf-id-42 |

cid | authority-id MUST be a CID assigned and registered by the [IEEE Registration Authority](https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries) to the entity responsible for this Endpoint. authority-id MUST use hex encoding of the 24-bit ID (resulting in 6 hex characters).instance-id syntax is defined by this entity, who is also responsible for determining instance-id assignment mechanisms and for ensuring uniqueness of the instance-id within the context of the CID.Example: cid:3AA3F8:my-unique-usp-id-42 |  
pen | authority-id MUST be a Private Enterprise Number assigned and registered by the [IANA](http://pen.iana.org/pen/PenApplication.page) to the entity responsible for this Endpoint. authority-id MUST use decimal encoding of the IANA-assigned number.instance-id syntax is defined by this entity, who is also responsible for determining instance-id assignment mechanisms and for ensuring uniqueness of the instance-id within the context of the Private Enterprise Number.Example: pen:3561:my-unique-bbf-id-42|  
self | An authority-id for "self" MUST be between 0 and 6 non-reserved characters in length. When authority-id is 1 or more characters, it is generated by the Endpoint.The Endpoint ID, including instance-id, is generated by the Endpoint.The Endpoint MUST change its Endpoint ID if it ever encounters another Endpoint using the identical Endpoint ID.Example: self::my-Agent |  
user | An authority-id for "user" MUST be between 0 and 6 non-reserved characters in length.The Endpoint ID, including instance-id, is assigned to the Endpoint via a user or management interface. |  
os | authority-id MUST be zero-length.instance-idis <OUI> "-"<SerialNumber>, as defined in TR-069[2](https://www.broadband-forum.org/technical/download/TR-069.pdf), Section 3.4.4. Example: os::00256D-0123456789 |  
ops | authority-id MUST be zero-length.instance-id is <OUI> "-" <ProductClass> "-" <SerialNumber>, as defined in [TR-069](https://www.broadband-forum.org/technical/download/TR-069.pdf), Section 3.4.4.Example: ops::00256D-STB-0123456789 |  
uuid | authority-id MUST be zero-length.instance-id is a [UUID](https://tools.ietf.org/html/rfc4122)Example:uuid::f81d4fae-7dec-11d0-a765-00a0c91e6bf6  
imei | authority-id MUST be zero-length.instance-id is an IMEI as defined by GSMA(https://imeidb.gsma.com/imei/index).Example: imei::990000862471854 |  
proto | authority-id MUST be between 0 and 6 non-reserved characters (except ".") in length."proto" is used for prototyping purposes only. Any authority-id and instance-id value (or scheme for creating the value) is left to the prototyper.Example: proto::my-Agent |  
doc | authority-id MUST be between 0 and 6 non-reserved characters in length."doc" is used for documentation purposes only (for creating examples in slide decks, tutorials, and other explanatory documents). Any authority-id and instance-id value (or scheme for creating the value) is left to the document creator. |

**R-ARC.3** - BBF OUI (00256D) and Private Enterprise Number (3561) are reserved for use in BBF documentation and BBF prototyping and MUST NOT be used by any entity other than BBF.

**R-ARC.4** - The "proto" and "doc" authority-scheme values MUST NOT be used in production environments.

The "proto" and "doc" values are intended only for prototyping and documentation (tutorials, examples, etc.), respectively.

#### instance-id

**R-ARC.5** - instance-id MUST be encoded using only the following characters:

instance-id = unreserved / pct-encoded  
 unreserved = ALPHA / DIGIT / "-" / "." / "\_"  
 pct-encoded = "%" HEXDIG HEXDIG

The above expression uses the Augmented Backus-Naur Form (ABNF) notation of [RFC2234](https://www.ietf.org/rfc/rfc2234.txt), including the following core ABNF syntax rules defined by that specification: ALPHA (letters), DIGIT (decimal digits), HEXDIG (hexadecimal). It is taken from [RFC3986](https://www.ietf.org/rfc/rfc3986.txt) as the set of unreserved characters and percent-encoded characters that are acceptable for all components of a URI. This set is also allowed for use in URNs [RFC2141](https://www.ietf.org/rfc/rfc2141.txt), and all MTP headers.

**R-ARC.6** - An instance-id value MUST be no more than 50 characters in length.

Shorter values are preferred, as end users could be exposed to Endpoint IDs. Long values tend to create a poor user experience when users are exposed to them.

## Service Elements

"Service Element" is a general term referring to the set of Objects, sub-Objects, commands, events, and parameters that comprise a set of functionality that is manipulated by a Controller on an Agent. An Agent’s Service Elements are represented in a Data Model - the data model representing an Agent’s current state is referred to as its Instantiated Data Model, and the data model representing the Service Elements it supports is called its Supported Data Model. The Supported Data Model is described in a Device Type Definition (DT). An Agent’s Data Model is referenced using Path Names.

### Data Models

USP is designed to allow a Controller to manipulate Service Elements on an Agent using a standardized description of those Service Elements. This standardized description is known as an information model, and an information model that is further specified for use in a particular protocol is known as a “Data Model”.

*Note: This should be understood by those familiar with CWMP. For those unfamiliar with that protocol, a Data Model is similar to a Management Information Base (MIB) used in the Simple Network Management Protocol (SNMP) or YANG definitions used in NETCONF.*

This version of the specification defines support for the following Data Model(s):

* The Device:2 Data Model for CWMP ([1](https://github.com/BroadbandForum/usp/tree/master/data-model))

This Data Model is specified in XML. The schema and normative requirements for defining Objects, Parameters, Events, and Commands for the Device:2 Data Model for [CWMP](https://github.com/BroadbandForum/usp/tree/master/data-model), and for creating Device Type Definitions based on that Data Model, are defined in [Broadband Forum TR-106, “Data Model Template for TR-069 Enabled Devices”](https://www.broadband-forum.org/technical/download/TR-106_Amendment-8.pdf).

The use of USP with any of the above data models creates some dependencies on specific Objects and Parameters that must be included for base functionality.

#### Instantiated Data Model

An Agent’s Instantiated Data Model represents the Service Elements (and their state) that are currently represented by the Agent. The Instantiated Data Model includes a set of Objects, and the sub-Objects (“children”), Parameters, Events, and Commands associated with those objects.

#### Supported Data Model

An Agent’s Support Data Model represents the Service Elements that an Agent understands. It includes references to the Data Model(s) that define the Objects, Parameters, Events, and Commands implemented by the Service Elements the Agent represents. A Supported Data Model consists of the union of all Device Type Definitions used by the Agent.

#### Objects

Objects are data structures that are defined by their sub-Objects, Parameters, Events, Commands, and creation criteria. They are used to model resources represented by the Agent. Objects may be static (single-instance) or dynamic (a multi-instance Object, or “table”).

##### Single-Instance Objects

Static Objects, or “single instance” Objects, are not tables and do not have more than one instance of them in the Agent. They are usually used to group Service Element functionality together to allow for easy definition and addressing.

##### Multi-Instance Objects

Dynamic Objects, or “multi-instance” Objects, are those Objects that can be the subject of “create” and “delete” operations (using the Add and Delete messages, respectively), with each instance of the Object represented in the Instantiated Data Model with an Instance Identifier (see below). A Multi-Instance Object is also referred to as a “Table”, with each instance of the Object referred to as a “Row”. Multi-Instance Objects can be also the subject of a [search][#search].

#### Parameters

Parameters define the attributes or variables of an Object. They are retrieved by a Controller using the read operations of USP and configured using the update operations of USP (the Get and Set messages, respectively). Parameters have data types and are used to store values.

#### Commands

Commands define Object specific methods within the Data Model. A Controller can invoke these methods using the “Operate” message in USP (i.e., the Operate message). Commands have associated input and output arguments that are defined in the Data Model and used when the method is invoked and returned.

#### Events

Events define Object specific notifications within the Data Model. A Controller can subscribe to these events by creating instances of the Subscription table, which are then sent in a [Notify Request by the Agent](/specification/messages/notify/). Events may also have information associated with them that are delivered in the Notify Request - this information is defined with the Event in the Data Model.

#### Path Names

A Path Name is a fully qualified reference to an Object, Object Instance, or Parameter in an Agent’s instantiated or Supported Data Model. The syntax for Path Names is defined in [TR-106](https://www.broadband-forum.org/technical/download/TR-106_Amendment-8.pdf).

**R-ARC.7** - All USP endpoints MUST support the Path Name syntax as defined in [TR-106](https://www.broadband-forum.org/technical/download/TR-106_Amendment-8.pdf).

Path Names are represented by a hierarchy of Objects (“parents”) and sub-Objects (“children”), separated by the dot “.” character, ending with a parameter if referencing a parameter path. There are four different types of Path Names used to address the data model of an Agent:

1. Object Path - This is a Path Name of either a single-instance (“static”) Object, or the Path Name to a Data Model Table (i.e., a Multi-Instance Object). An Object Path ends in a “.” Character (as specified in [TR-106](https://www.broadband-forum.org/technical/download/TR-106_Amendment-8.pdf)), except when used in a [reference parameter](#reference_following). When addressing a Table in the Agent’s Supported Data Model that contains one or more Multi-Instance Objects in the Path Name, the sequence “{i}” is used as a placeholder (see the [GetSupportedDM message](/specification/messages/getsupporteddm/)).
2. Object Instance Path - This is a Path Name to a Row in a Table in the Agent’s Instantiated Data Model (i.e., an Instance of a Multi-Instance Object). It uses an Instance Identifier to address a particular Instance of the Object. An Object Instance Path ends in a “.” Character (as specified in [TR-106](https://www.broadband-forum.org/technical/download/TR-106_Amendment-8.pdf)), except when used in a [reference parameter](#reference_following).
3. Parameter Path - This is a Path Name of a particular Parameter of an Object.
4. Command Path - This is a Path Name of an Object defined [Operation](#operation_command_path_names).
5. Event Path - This is a Path Name of an Object defined [Event](#event_path_names).
6. Search Path - This is a Path Name that contains search criteria for addressing a set of Multi-Instance Objects and/or their Parameters. A Search Path may contain a Search Expression or Wildcard.

This creates two functions of Path Names: Addressing and Searching. The first five paths are used for addressing a particular Object, Parameter, Command, or Event. A Search Path uses Searching to return a set of Object Instances and/or their Parameters. When addressing, the expectation is that the Path Name will resolve to either 0 or 1 instance (and depending on the context, 0 instances could be an error). When searching, the expectation is that the Search Path will resolve to 0, 1, or many instances (and depending on the context, 0 instances is often not an error).

*NOTE: When resolving a Path Name, the Agent is expected to use locally cached information and/or information that can be obtained rapidly and cheaply. Specifically, there is no expectation that the Agent would issue a network request in order to resolve a Path Name.*

*NOTE: Obviously only one form of addressing or searching can be used for a given Instance Identifier in a Path Name, but different forms of addressing can be used if more than one Instance Identifier needs to be specified in a Path Name.*

For example, the following Path Name uses Unique Key Addressing for the Interface table but a Search Expression for the Ipv4Address table to select Enabled Ipv4 Addresses associated with the ”eth0” IP Interface:

.Interface.[Name==“eth0”].Ipv4Address.{addr}.IPAddres::{addr.Status==“Enabled”}

#### Relative Paths

Several USP messages make use of relative paths to address Objects or Parameters. A relative path is used to address the child Objects and parameters of a given Object Path or Object Instance Path. To build a Path Name using a Relative Path, a USP endpoint uses a specified Object Path or Object Instance Path, and concatenates the Relative Path. This allows some efficiency in Requests and Responses when passing large numbers of repetitive Path Names. This relative path may include [instance identifiers](#using_instance_identifiers_in_path_names) to Multi-Instance Objects.

For example, for an Object Path of:

Device.Wifi.Radio.1.

Relative paths would include parameters:

Status

SupportedStandards

OperatingStandards

Etc., as well as the following sub-Object and its parameters:

Stats.BytesSent

Stats.BytesReceived

Etc.

#### Using Instance Identifiers in Path Names

##### Addressing by Instance Number

Instance Number Addressing allows an Object Instance to be addressed by using its Instance Number in the Path Name. An Instance Number is expressed in the Path Name as a positive integer (>=1) with no additional surrounding characters. The Instance Number assigned by the Agent is arbitrary.

**R-ARC.8** - The assigned Instance Number MUST persist unchanged until the Object Instance is subsequently deleted (either by the USP Delete message or through some external mechanism). This implies that the Instance Number MUST persist across a reboot of the Agent, and that the Agent MUST NOT allow the Instance Number of an existing Object Instance to be modified by an external source.

For example, the .Interface table with an Instance Number of 3 would be addressed with the following Path Name: Device.IP.Interface.3.

##### Addressing by Unique Key

Key-based addressing allows an Object Instance to be addressed by using a Unique Key (as defined in [Device:2](https://github.com/BroadbandForum/usp/tree/master/data-model)) in the Path Name. This is possible since once a Parameter that is part of a unique key has its value set, then that value is immutable for the life of the Object that contains the Parameter.

For example, the Device.IP.Interface table has 2 separate unique keys; Name and Alias.

Unique Keys used for addressing are expressed in the Path Name by using square brackets surrounding a string that contains the name and value of the Unique Key parameter using the equivalence operator (==).

If an Object has a compound unique key (multiple parameters included within the same unique key), then all keys must be present in the Instance Identifier and separated by a comma (,) character (the order of the parameters does not have to follow the order of the parameters as defined in the unique key element as defined in [Device:2](https://github.com/BroadbandForum/usp/tree/master/data-model)).

For example, the Device.NAT.PortMapping table has a compound unique key consisting of RemoteHost, ExternalPort, and Protocol, which would be addressed with the following Path Name:

Device.NAT.PortMapping.[RemoteHost==““,ExternalPort==0,Protocol==“TCP”].

##### Searching with Expressions

Searching is a means of matching 0, 1 or many instances of a Multi-Instance Object by using the properties of Object. Search Paths use an Expression Variable enclosed in curly braces as the Instance Identifier within a Path Name and then appends a “::” to the end of the Path Name, followed by an Expression enclosed in another set of curly braces.

**R-ARC.9** - An Agent MUST return Path Names that include all Object Instances that match the criteria of a given Search Path.

**R-ARC.10** - An Expression Variable MUST be a valid identifier, which means that it MUST follow the same syntax as used for naming Parameters as defined in [TR-106](https://www.broadband-forum.org/technical/download/TR-106_Amendment-8.pdf).

The basic format of a Search Path is:

Device.IP.Interface.{<expression variable>}.Status::{<expression>}

An Expression consists of one or more Expression Components that are separated by the AND (&&) logical operator (NOTE: the OR logical operator is not supported).

The basic format of a Search Path with the Expression element expanded is:

Device.IP.Interface.{<expression variable>}.Status::{<expression component>&&<expression component>}

An Expression Component is a combination of an Expression Parameter followed by an Expression Operator followed by an Expression Constant.

The basic format of a Search Path with the Expression Component element expanded is:

Device.IP.Interface.{<expression variable>}.Status::{<expression parameter><expression operator><expression constant>}

An Expression Parameter is based on an Expression Variable found in the Path Name, which allows the Expression Component to represent a relative path based on where the Expression Variable is located in the Path Name.

For example, Device.IP.Interface.{intf}.Ipv4Address.{addr}.IPAddress::{…} means that “intf” inside the Expression Parameter represents the instances of the Device.IP.Interface.{i} Object whereas “addr” inside the Expression Parameter represents the instances of the Device.IP.Interface.{i}.Ipv4Address.{i} Object.

Further, this relative path can’t include any child tables. *(NOTE: this is never necessary because any child tables that need to be referenced in the Expression can and should have their own Expression Variables)*

An Expression Operator dictates how the Expression Component will be evaluated. The supported operators include: equals (==), not equals (!=), less than (<), greater than (>), less than or equal (<=), and greater than or equal (>=).

An Expression Parameter will always be of the type defined in the data model. Expression operators will only evaluate for appropriate data types. The literal value representations for all data types are found in [TR-106](https://www.broadband-forum.org/technical/download/TR-106_Amendment-8.pdf). **For string, boolean and enumeration types, only the '==' and '!=' operators are valid.**

The Expression Constant is the value that the Expression Component is being evaluated against; Expression Components must match the type as defined for the associated Parameter in [TR-181](https://github.com/BroadbandForum/usp/tree/master/data-model).

*NOTE: String values are enclosed in double quotes. In order to allow a string value to contain double quotes, quote characters can be percent-escaped as %22 (double quote). Therefore, a literal percent character has to be quoted as %25.*

Expressed as a [Backus-Naur Form (BNF)](https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form) for context-free grammars, the Search Expression lexical rules for referencing the Supported Data Model are:

[1] sdmpath ::= objpath | sdmparampath  
[2] sdmparampath::= sdmcomps '.' name  
[3] objpath ::= sdmcomps '.'  
[4] sdmcomps ::= name ( '.' sdmcomp )\*  
[5] sdmcomp ::= name | sdminst  
[6] sdminst ::= posnum | '{i}'  
[7] name ::= [A-Za-z\_] [A-Za-z\_0-9]\*  
[8] posnum ::= [1-9] [0-9]\*

The Search Expression lexical rules for referencing the Instantiated Data Model are:

[1] idmpath ::= objinstpath | parampath | cmdpath | evntpath | searchpath  
[2] objinstpath ::= comps '.' expr?  
[3] parampath ::= comps '.' namemod  
[4] cmdpath ::= comps '()' expr?  
[5] evntpath ::= comps '!' expr?  
[6] searchpath ::= comps expr  
[7] comps ::= name ( '.' comp )\*  
[8] comp ::= namemod | inst  
[9] inst ::= posnum | keyref | exprvar | '\*'  
[10] keyref ::= '[' keyexpr ( ',' keyexpr )\* ']'  
[11] keyexpr ::= relpath '==' value  
[12] exprvar ::= '{' name '}'  
[13] expr ::= '::' '{' exprcomp ( '&&' exprcomp )\* '}'  
[14] exprcomp ::= relpath oper value  
[15] relpath ::= name ( '.' namemod )+  
[16] namemod ::= name ( '#' elem )? '+'?  
[17] oper ::= '==' | '!=' | '<' | '>' | '<=' | '>='  
[18] elem ::= posnum | '\*'  
[19] value ::= literal | number  
[20] name ::= [A-Za-z\_] [A-Za-z\_0-9]\*  
[21] literal ::= '"' [^"]\* '"'  
[22] posnum ::= [1-9] [0-9]\*  
[23] number ::= '0' | ( '-'? posnum )

##### Search Examples

Valid Searches:

Status for all IP Interfaces with a “Normal” type:

Device.IP.Interface.{intf}.Status::{intf.Type==“Normal”}

Ipv4 Addresses for all IP Interfaces with a Normal type and a Static addressing type:

Device.IP.Interface.{intf}.Ipv4Address.{addr}.IPAddress::{intf.Type==“Normal”&&addr.AddressingType==“Static”}

Ipv4 Addresses for all IP Interfaces with a Normal type and Static addressing type that have at least 1 Error Sent

Device.IP.Interface.{intf}.Ipv4Address.{addr}.IPAddress::{intf.Type==“Normal”&&intf.Stats.ErrorsSent>0&&addr.AddressingType==“Static”}

Searches that are NOT VALID::

Invalid because the Expression Component is missing:

Device.IP.Interface.{intf}.

Invalid because the Expression Component is empty:

Device.IP.Interface.{intf}.::{}

Invalid because the Expression Component has an Expression Parameter that descends into a child table (always need to use a separate Expression Variable for each child table instance)

Device.IP.Interface.{intf}.Status::{intf.Type==“Normal”&&intf.Ipv4Address.\*.AddressingType==“Static”}

#### Searching by Wildcard

Wildcard-based searching is a means of matching all currently existing Instances (whether that be 0, 1 or many instances) of a Multi-Instance Object by using a wildcard character “\*” in place of the Instance Identifier.

**R-ARC.11** - An Agent MUST return Path Names that include all Object Instances that are matched by the use of a Wildcard.

Examples:

All parameters for all IP Interfaces that currently exist

Device.IP.Interface.\*.

Type of each IP Interface that currently exists

Device.IP.Interface.\*.Type

#### Other Path Decorators

##### Reference Following

[Device:2](https://github.com/BroadbandForum/usp/tree/master/data-model) contains Parameters that reference other Parameters or Objects. The Reference Following mechanism allows references to Objects (not Parameters) to be followed from inside a single Path Name. Reference Following is indicated by a “+” character after the name of the Parameter that is referencing the Object followed by a “.”, followed by Objects or Parameters that are children of the Referenced Object.

For example, Device.NAT.PortMapping.{i}.Interface references an IP Interface Object (Device.IP.Interface.{i}.) and that Object has a Parameter called “Name”. With Reference Following, a Path Name of Device.NAT.PortMapping.1.Interface+.Name references the “Name” Parameter of the Interface Object that the PortMapping is associated with (i.e. it is the equivalent of using Device.IP.Interface.1.Name as the Path Name.

The steps that are executed by the Agent when following the reference in this example would be:

1. Retrieve the appropriate instance of the PortMapping Object based on the Instance Number Addressing information
2. Retrieve the value of the reference parameter Parameter that contains the reference, Interface, which in this case has the value “Device.IP.Interface.1”
3. Replace the preceding path (Device.NAT.PortMapping.1.Interface+) with the value retrieved in Step 2
4. Append the remainder of the Path Name (.Name), which builds the Path Name: Device.IP.Interface.1.Name
5. Use Device.IP.Interface.1.Name as the Path Name for the action

\*Note: It should be noted that according to the [Device:2 Schema](https://www.broadband-forum.org/technical/download/TR-106_Amendment-8.pdf), reference parameters:

* Always contain Path Names (not Search Expressions)
* When configured, can be configured using Path Names using Instance Number Addressing or Unique-Key Addressing, however:
* When the value of a reference parameter is read, all Instance Identifiers are returned as Instance Numbers.\*

**R-ARCH.12** - A USP Agent MUST support the ability to use Key-based addressing (as described in #REF) in reference values.

For example, the following paths might illustrate a reference to the same object (defined as having the KeyParam parameter as unique key) instance using an Instance Number and then a key value:

* Object.SomeReferenceParameter = “Object.FooObject.5”
* Object.SomeReferenceParameter = “Object.FooObject.[KeyParam==”KeyValueForInstance5”]”

In the first example, the reference points to the FooObject with Instance Number 5. In the second example, the reference points to the FooObject with a KeyParam value of “KeyValueForInstance5”.

**R-ARCH.13** - The following requirements relate to reference types and the associated Agent behavior:

* An Agent MUST reject an attempt to set a strong reference parameter if the new value does not reference an existing parameter or object.
* An Agent MUST NOT reject an attempt to set a weak reference parameter because the new value does not reference an existing parameter or object.
* An Agent MUST change the value of a non-list-valued strong reference parameter to a null reference when a referenced parameter or object is deleted.
* An Agent MUST remove the corresponding list item from a list-valued strong reference parameter when a referenced parameter or object is deleted.
* An Agent MUST NOT change the value of a weak reference parameter when a referenced parameter or object is deleted.

##### List of References

The USP data models have Parameters whose values contain a list of references to other Parameters or Objects. This section explains how the Reference Following mechanism allows those references to be followed from inside a single Path Name. The Reference Following syntax as defined above still applies, but it is preceded by a means of referencing a specific instance within the list. The additional syntax consists of a “#” character followed by list item number (1-indexed), which is placed between the name of the Parameter that contains the list of references and the “+” that indicates that the reference should be followed. To follow *all* references in the list, the endpoint can specify a "#" character followed by a wildcard ("\*") character and the "+" character to follow the reference (i.e., "ReferenceParameter#\*+").

For example, Device.WiFi.SSID.{i}.LowerLayers references a list of WiFi Radio Object (defined as Device.WiFi.Radio.{i}.) Instances that are associated with the SSID. This Object has a Name Parameter; so when following the first reference in the list of references a Path Name of Device.WiFi.SSID.1.LowerLayers#1+.Name references the Name of the WiFi Radio associated with this SSID Object Instance.

The steps that are executed by the Agent when following the reference in this example would be:

1. Retrieve the appropriate Device.Wifi.SSID.{i} instance based on the Instance Number Addressing information
2. Retrieve the value of the LowerLayers Parameter, which in this case has a value of “Device.WiFi.Radio.1, Device.WiFi.Radio.2”
3. Retrieve the first list item within the value retrieved in Step 2 (i.e., “Device.WiFi.Radio.1”)
4. Replace the preceding path (Device.WiFi.SSID.1.LowerLayers#1+) with the value retrieved in Step 3
5. Append the remainder of the Path Name (.Name), resulting in a Path Name of: Device.WiFi.Radio.1.Name
6. Use Device.WiFi.Radio.1.Name as the Path Name for the action

##### Search Expressions and Reference Following

The Reference Following and Search Expression mechanisms can be combined.

For example, reference the Signal Strength of all WiFi Associated Devices using the “ac” Operating Standard on the “MyHome” SSID, you would use the Path Name:

Device.WiFi.AccessPoint.{ap}.AssociatedDevice.{dev}.SignalStrength:: {ap.SSIDReference+.SSID==“MyHome”&&dev.OperatingStandard==“ac”}

##### Operations/Commands

The [Operate message](/messages/operate/) allows a USP Controller to execute Commands defined in the USP data models. Commands are synchronous or asynchronous operations that don’t fall into the typical REST-based concepts of CRUD-N that have been incorporated into the protocol as specific messages. Commands are addressed like Parameter Paths that end with parentheses “()” to symbolize that it is a Command.

For example: Device.IP.Interface.[Name==“eth0”].Reset()

##### Events

The Notify request allows a type of generic event (called Event) message that allows a USP Agent to emit events defined in the USP data models. Events are defined in and related to Objects in the USP data models like commands. Events are addressed like Parameter Paths that end with an exclamation point “!” to symbolize that it is an Event.

For example: Device.LocalAgent.Boot!

[<-- Overview](/specification/) [Discovery -->](/specification/discovery/)